(4)

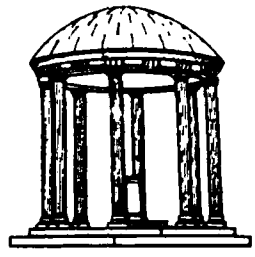
The Penalty of Context-Switch Time in Distributed Computing

TR88-025

May 1988

Contract N00014-86-K-0680

Mark C. Davis
Bill O. Gallmeister



The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

The Penalty of Context-Switch Time in Distributed Computing

Mark C. Davis
Bill O. Gallmeister

Department of Computer Science *
University of North Carolina
Chapel Hill, NC 27599-3175

May 13, 1988

Abstract

Context-switch time is a significant cost in distributed computing, affecting throughput and response time. We report statistics gathered for a large network of Sun 2's, Sun 3's and DEC VAX computers.

1 Introduction and Definition of Components

Context switching plays a bigger part of the cost of computing as multitasking becomes more useful and widespread. One application that depends heavily on multitasking is distributed computing.

A context switch is the act of a computer changing from execution of one task to another task. During such a switch, the computer must save any state in the Central Processing Unit (CPU) or the Memory Management Unit (MMU) which might be modified by the next task.

We were interested in the effect and frequency of context switching in what we consider a modern computing environment: a large number of diskless workstations and fileservers connected to an Ethernet. In our case, we had about 60 diskless Sun 2's and 3's served by 14 Sun 2, Sun 3, VAX 11/750 and 11/780 computers with disks. Much of the network traffic generated by these machines was to service the Network File System (NFS), which was built on the Remote Procedure Call (RPC) protocol, both developed by Sun Microsystems[2].

*This Research was supported by Office of Naval Research Contract N00014-86-K-0680.

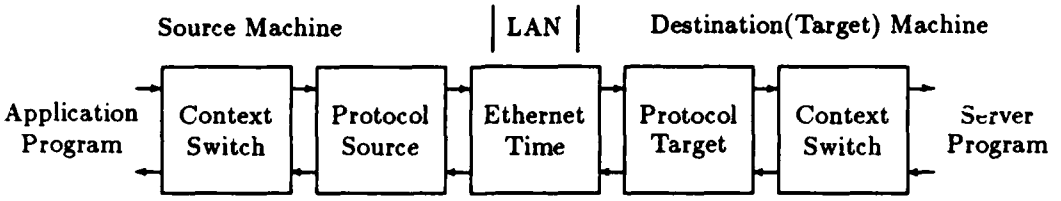


Figure 1: Components of a Two-Way Network Transmission

RPC provided support for two way service calls, and the NFS calls did involve two way communication, so this is the type of network traffic we analyzed. The network topology also involved an IBX Integrated Voice/Data Switch that bridged 6 Ethernet SubLAN's. As part of our measurements, we unavoidably determined the overhead involved in using this bridge.

2 A Network Transaction Requires Many Steps

Context switching was only one of several items that consumed time during a network transaction. In a UNIX environment running on Sun 2 and 3 computers and on DEC VAX computers, a simple request sent over the network had to go through several stages, as shown in Figure 1. The application program caused a context switch so that the protocol program could deal with the data and interface with the Ethernet controller. Then, the message was passed over the Ethernet, incurring some delays in the controllers at each end as well as the time required to transmit the data over the wiring. The message was processed on the target machine by another protocol program. When it had identified the server program, it caused a context switch to the server program so that it could process the data. The result of the service then progressed back through the same steps to the application program on the source machine. In summary, servicing this request required four context switches, source protocol, target protocol, and Ethernet transmission time. There were some special cases: If the source and target machines were both the same Sun 2 or Sun 3, there was no Ethernet transmission and only two context switches were required instead of four. (Interestingly, the DEC VAXes still transmitted the packets on the Ethernet and incurred all four context switches.)

3 Measurement Methodology

Since we wanted to analyze the impact of context switch time, we needed accurate performance measurements. Because of our interest in large numbers of active processes and the special characteristics of the Sun MMU, we were unable to use previous context switch benchmarks [1]. We also gathered statistics on numbers of context switches and number of active processes on various computers. To measure the network transmission time, we wrote a program to use the RPC protocol.

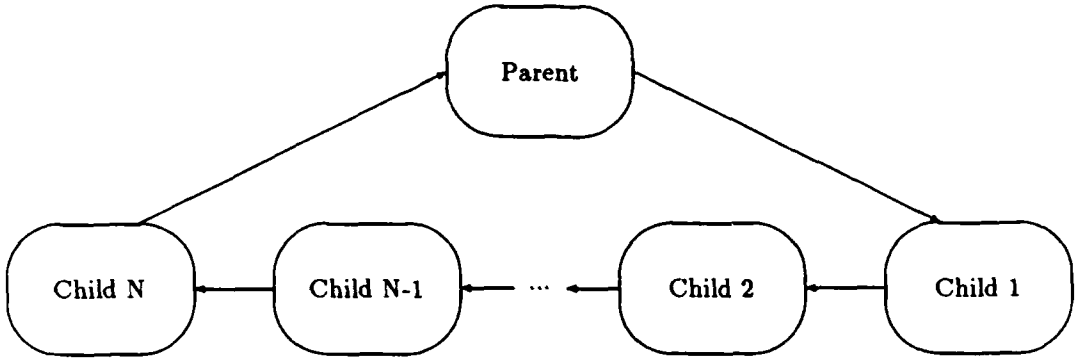


Figure 2: Ring of Communicating Processes

3.1 Context Switch Benchmark Program

The Sun MMU had the ability to store information for up to eight contexts, with each context being equivalent to a process. This feature implied that the context-switch performance would change as the number of active processes changed. Previous benchmark programs had only used two processes to benchmark the context-switch time. The number of processes to use in our benchmark was variable. The main program created new processes until enough processes existed. Each child process was aware of the process identification number (PID) of the previous sibling; the first child was aware of the parent's PID and the parent was aware of the PID for the last child created. This formed a ring (Figure 2) of processes which could communicate via the UNIX **signal** facility. Each process was initialized to handle a particular signal by immediately signaling its sibling. The children processes all went to sleep. The original program used the **kill** system call to wake up the previous process, and then went to sleep. The result was that one process was awake at any given time, processing a signal, and this "control" moved around the ring of processes. Because all processes were sleeping except for the one currently processing the signal, control proceeded around the ring at the maximum rate possible. Figure 2 shows the arrangement of the processes and the signaling paths. When the children had been signaled the appropriate number of times, they terminated. The original program then calculated and printed the statistics for the run.

3.2 Number of Processes and Switching Frequency

The number of processes and the context-switch rate were determined for 50 machines of various types attached to the network. We used UNIX commands and special routines to directly read kernel memory to obtain CPU time and number of context switches. These programs were run at two hour intervals during weekday operating periods to obtain representative data. The number of context switches was normalized to non-idle CPU time.

Machine Type	Context Switch Time
VAX 11/789	1
VAX 11/750	2
Sun 4	.5 - .85
Sun 3	.8 - 2.5
Sun 2	1.7 - 6

Table 1: Context Switch Times (in msec)

3.3 RPC Call Benchmark Program

The objective of the RPC call benchmark program was to determine the best case time for common network transactions. As noted above, one common type of transmission is the two-way RPC call. To measure the total network transaction time, we used null procedure calls built in to the NFS servers and YPBIND, another server program that runs on diskless Sun computers. By using the null procedure call, no work was done by the server. The application program did no work to except to call the server, so the resulting time was indicative of the overhead of common network transactions. The test program took as parameters the name of the target machine and the number of calls to make to the target. The lowest levels of RPC programming [2] were used to eliminate unnecessary overheard. For example, the name of the target machine and the port to use were identified before the timing loop was started. To get the best case performance, we ran several tests of 10,000 calls each during low activity periods. Data was collected on several occasions to ensure that interfering network traffic did not affect the results.

4 The Results - Effect of Context Switch Time

4.1 General Purpose Unix Frequency and Cost

The observed context-switching times for several types of machines are shown in Table 1. As expected, the context-switch time for the Sun workstations varied according to the number of active processes. The first number listed is an average when 6 or less processes were running. The second number is an average when 9 or more processes were running. There was a sharp increase in context-switch time between 6, 7, and 8 processes, but little incremental increase when more that 9 processes were running. Under the conditions that we observed, usually fewer than 6 processes were ready to run on the Suns, so we used the lower numbers in our calculations.

Table 2 shows the number of active processes and the frequency of context switches for several types of machines during normal operations. A surprising number of processes and context switches were measured. Note that even for the machine with the most users, at least three processes were needed for each user. The context switch times have been normalized

Machine Type	Typical Number of Active Processes	Context Switches per Second CPU Time
VAX 11/785 (60 users)	250	40
Utility VAX 11/750	50	35
Sun 3 Client	25-50	50-600
Sun 3 Server	35	200

Table 2: Processes and Context Switch Rates

From a Sun 3/75	Self (47%)	Local Sun 3	Distant Sun 3	Distant Sun 2	VAX 750	VAX 780
Protocol Source	*2.22*	2.22	2.22	2.22	2.22	2.22
Protocol Target	*2.60*	2.60	2.60	*7.83*	*5.88*	*6.97*
Context Switch Source	0.80	1.60	1.60	1.60	1.60	1.60
Context Switch Target	0.80	1.60	1.60	3.40	4 00	2.00
Ethernet Time		*0.82*	0.82	0.82	0.82	0.82
IBX Switch Time			*7.60*	7.60		7.60
Measured Total Time	6.42	8.90	16.44	23.47	14.59	25.81

Table 3: RPC Components (in msec)

to non-idle cpu time. In the case of the multiuser machine, the cpu was used almost 100% of the time. For the other machines, percent of cpu varied widely and was loosely related to the wide variations in context switch rate.

4.2 Remote Procedure Calls and the Cost of Context Switches

Our remote procedure call benchmark was ported to Sun 2's, Sun 3's, VAX 11/750's and VAX 11/780's. The machine of most interest was the Sun 3, and Table 3 displays these results. Since the Sun does not use the Ethernet controller while making RPC calls to itself, we were able to use this data to determine Ethernet delay times and IBX bridge delay times. The items calculated from each test are *emphasized.*

Table 4 show the results when other types of machines were used. Since the Ethernet controller time could not be measured for the VAX, we assumed that the time would be the same as for the Sun. The results of these tests agree with results from the previous test from the Sun 3/75.

Source Machine	Sun 2	VAX 780	VAX 780
Target Machine	Self	Self	750
Percent source	47%	53%	
Protocol Source	*5.09*	*6.59*	6.59
Protocol Target	*5.96*	*4.28*	*17.86*
Context Switch Source	1.70	2.00	4.00
Context Switch Target	1.70	2.00	2.00
Ethernet Time		0.82	0.87
IBX Switch Time			7.54
Measured Total Time	14.45	16.22	40.45

Table 4: RPC Components (in msec) from other machines

	Target			
Source	Sun 3	Sun 2	VAX 11/750	VAX 11/780
Sun 3	36.20	31.51	38.57	26.47
Sun 2	31.67	36.34	33.17	27.41
VAX 11/780	24.90	18.62	19.70	24.66

Table 5: Percent of Context Switch Time For RPC

5 Summary and Conclusions

As can be seen from Table 5, context switching can constitute from 10% to 40% of the overhead of network transmissions. Since multitasking has undeniable advantages for distributed computing, future computer architectures and operating systems must place more emphasis on fast context switching.

References

[1] Jerome Feder. The Evolution of Unix System Performance. *AT&T Bell Laboratories Technical Journal*, 63(8):1791-1814, October 1984.

[2] Sun Microsystems. Remote Procedure Call Programming Guide. In *Networking on the Sun Workstation*, Mountain View, California, February 1986.